Contents

1 Question1

1.1 Advantages of Hadoop over Traditional Data Warehousing

Scalability: Traditional data warehouses are like that old-school closet in your house - there's only so much you can stuff into it. Hadoop, on the other hand, is like building an expandable storage unit. It scales massively and inexpensively because it's designed to run on low-cost commodity hardware. You need more storage? Just add more nodes. Easy-peasy.

Flexibility in Handling Data:Hadoop is like a culinary genius who can cook anything from gourmet to street food. It can handle all sorts of data - structured, unstructured, semi-structured. In contrast, traditional data warehouses are like picky eaters, primarily handling structured data and often hiccupping on anything else.

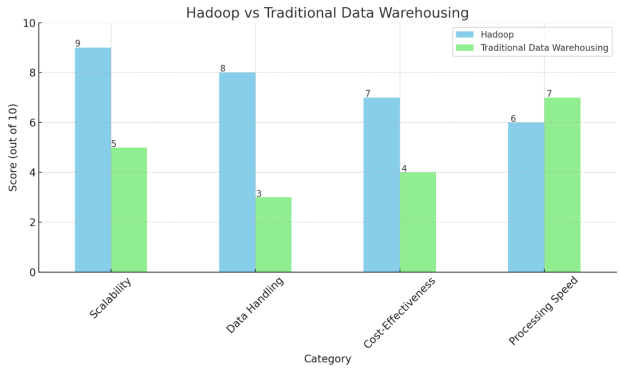

Figure 1: Q1.1

1.2 Reasons Not to Replace a Relational Database with Hive

Now, why wouldn't you ditch your reliable relational database for Hive? Here's why:

Transaction Support: Hive is like a historian, great at recording history, but not so hot at the day-to-day transactions. It's not built for real-time updates and deletes like a traditional relational database.

Latency: Hive is a bit like a sloth when it comes to speed for small, quick queries. Relational databases are more like cheetahs - fast and efficient for these tasks.

SQL Compliance and Features:Hive's SQL capabilities are growing but still can't match the rich, full-fledged SQL operations and functions offered by traditional databases. It's like comparing a Swiss Army knife to a specialized chef's knife.

1.3 CAP Properties for NoSQL Systems

In the NoSQL world, it's all about the CAP (Consistency, Availability, Partition tolerance) theorem. Choosing two is like picking your favorite ice cream flavors - you can't have them all:

CA (Consistency + Availability):Great for systems where you need rock-solid data accuracy and availability, but it's like having a party but not inviting the neighbors (partition tolerance).

CP (Consistency + Partition Tolerance):This is for scenarios where you can't afford to lose any data accuracy even if some parts of your system go down, like having a backup generator during a blackout.

AP (Availability + Partition Tolerance):Ideal when your system must always be up and running, even if some data might be a bit stale, like a 24/7 convenience store that might run out of fresh bread.

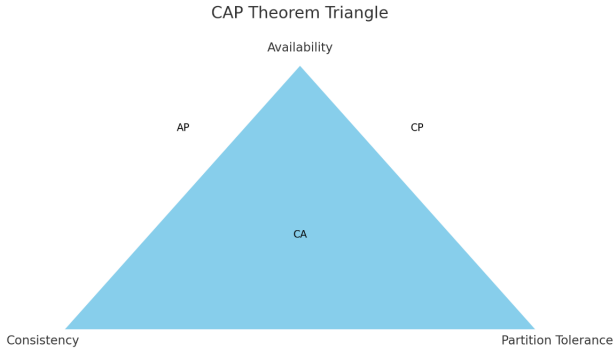

Figure 2: Q1.3.1

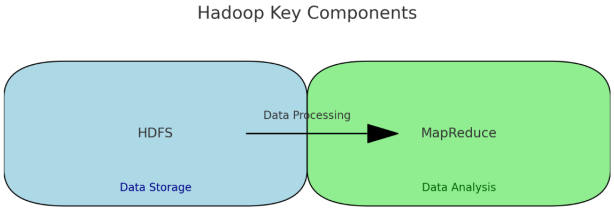

Figure 3: Q1.3.1

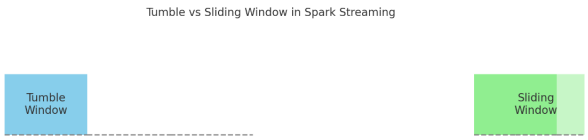

Figure 4: Q1.3.1

1.4 Horizontal Scaling

"Horizontal scaling," a term as fancy as it sounds, is simply about adding more machines (nodes) to your system, like hiring more chefs to handle a busy kitchen. It contrasts with vertical scaling, which is like upgrading your existing chef to a super-chef with better skills and tools.

1.5 Data Locality in Hadoop

Data locality in Hadoop is like having your tools right where you need them in a workshop. It moves computation to the data rather than moving data to the computation. This approach minimizes network congestion and increases the overall system performance, much like having all your ingredients on the counter speeds up cooking.

1.6 Key Components/Concepts of Hadoop

Hadoop's big guns are:

HDFS (Hadoop Distributed File System):Think of HDFS as a massive bookshelf, stretching infinitely, storing data across a cluster of machines.

MapReduce:This is the brain that processes the data. It's like a massive, parallel assembly line, breaking down tasks and working on them concurrently.

1.7 Tool for Importing Data into HDFS

To regularly scoop up a portion of your relational database and dump it into HDFS, Apache Sqoop is your go-to tool. It's like a diligent courier, transferring data efficiently. Why do this?

Offloading Processing: To reduce the load on your primary database, like giving your main chef a break by hiring a sous chef.

Big Data Analysis:To leverage Hadoop's power for heavy-duty data analysis, akin to moving from a home kitchen to a commercial one for big catering orders.

1.8 Join Type in Hive That Acts Like a Filter

In Hive, the "LEFT SEMI JOIN" is the unique one. It's more of a selective filter than a traditional join. It's like having a guest list at a party - only the ones on the list (in the left table and matching the right table) get in. It differs from inner and outer joins, which are more about mingling and expanding the guest list.

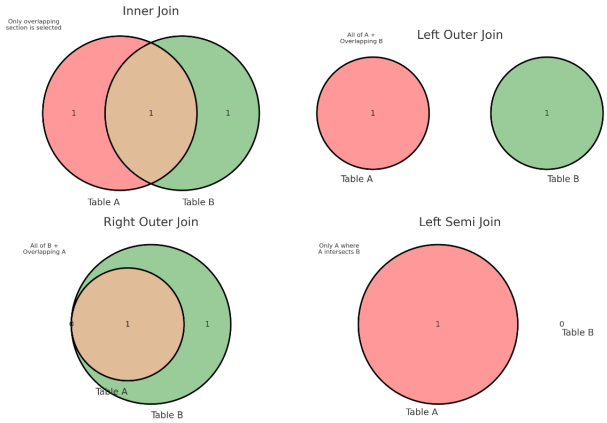

Figure 5: Q1.8

1.9 Complex Data Types Supported by Hive

Hive supports several complex data types, but the main three are: Arrays:Like a list of ingredients in a recipe. Maps:Think of a dictionary with word definitions. Structs:Similar to a record in a database, grouping related elements.

1.10 Reasons for Using Complex Data Types in Hive

Using these complex types in Hive is like adding spices to a dish: Handling Multi-Dimensional Data:Great for when data isn't just a flat line but has depth, like a lasagna. Simplifying Queries:It's like packing a suitcase neatly - easier to manage and access. Efficiency in Data Representation: They allow for a more accurate and efficient way to represent real-world data, like using a map instead of a long list of directions.

1.11 Output of Hive Query

For the Hive query "SELECT sentences(txt) FROM quotes˙table;" - assuming "txt" is a column of text (quotes) - it will split each quote into sentences. It's like taking a paragraph and neatly breaking it down into individual sentences.

1.12 Wordcount Performance: Spark vs. Hive

Spark outperforms Hive in wordcount because:

In-Memory Processing:Spark processes data in memory, like a chef who keeps all ingredients on the counter, whereas Hive reads from and writes to disk, like a chef who has to fetch everything from the pantry.

Optimized Execution Plans:Spark optimizes the workflow more effectively, planning its cooking steps better than Hive.

1.13 When Not to Use Spark SQL over HiveQL

Despite its speed, you might not want to use Spark SQL if:

Your Cluster is Overloaded:Spark's in-memory processing can be a burden if resources are tight, like having too many appliances running in a small kitchen.

Data Size is Manageable:For smaller datasets, Hive might be more cost-effective, like using a small frying pan instead of a large one for a single egg.

1.14 Tumble vs. Sliding Window in pySpark

In the world of streaming data, a "Tumble Window" is like taking snapshots at regular intervals (say, every hour), whereas a "Sliding Window" is like a rolling camera, capturing data over a rolling time frame (like the last 30 minutes, continuously).

In pySpark, these are implemented differently, with Tumble Window using functions like 'window' and 'groupBy', while Sliding Window might use 'window' with a slide duration specified.

1.15 Watermark in Spark Streaming

A "Watermark" in Spark streaming is like a bouncer at the club, deciding how late is too late for data to join the party. It's crucial for dealing with late data and ensuring system performance isn't bogged down by stragglers.

In a cheese warehouse, for example, it could be used to ignore temperature readings that are too old, ensuring the analysis reflects current conditions and not historical anomalies.

2 Question 2

2.1 SQL Query Sqoop Command

2.1.1 Sqoop Command to Import Data into HDFS:

```
literate
--move data from rds to hdfs
sqoop import --connect jdbc:mysql://'bigdata-final-db.cmwwild07s4i.us-east-1.rds.amazonaws.com'
    --username admin --Ct123778 --table question2 --target-dir /user/Hadoop/ads/
```

2.1.2 SQL Query to Load Data in RDS:

```
literate

-- create the table as already connect the rds on mysql workbench

--create data base
create database final;

--create tbale
CREATE TABLE IF NOT EXISTS question2 (
    id BIGINT,
    click TINYINT,
    hour VARCHAR(8),
    C1 INT,
    banner_pos INT,
    site_id VARCHAR(255),
    site_domain VARCHAR(255),
    site_category VARCHAR(255),
    app_id VARCHAR(255),
    app_domain VARCHAR(255),
    app_category VARCHAR(255),
    device_id VARCHAR(255),
    device_ip VARCHAR(255),
    device_model VARCHAR(255),
    device_type INT,
    device_conn_type INT,
```

```
    C14 INT,
    C15 INT,
    C16 INT,
    C17 INT,
    C18 INT,
    C19 INT,
    C20 INT,
    C21 INT
);

--load data
LOAD DATA INFILE '/user/Hadoop/ads/ads.csv'
INTO TABLE question2
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(id, click, hour, C1, banner_pos, site_id, site_domain, site_category, app_id, app_domain,
    app_category, device_id, device_ip, device_model, device_type, device_conn_type, C14, C15, C16,
    C17, C18, C19, C20, C21);
```

2.2 Create Hive Table:

```
literate
CREATE TABLE IF NOT EXISTS ads_lanston (
    id INT,
    click TINYINT,
    hour VARCHAR(8),
    C1 INT,
    banner_pos INT,
    site_id STRING,
    site_domain STRING,
    site_category STRING,
    app_id STRING,
    app_domain STRING,
    app_category STRING,
```

```
    device_id STRING,
    device_ip STRING,
    device_model STRING,
    device_type INT,
    device_conn_type INT,
    C14 INT,
    C15 INT,
    C16 INT,
    C17 INT,
    C18 INT,
    C19 INT,
    C20 INT,
    C21 INT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
LOAD DATA INPATH '/user/Hadoop/ads/ads.csv' INTO TABLE ads_lanston;
```

2.3 2.3

2.3.1 Load Data into Hive Table:

```
literate
--load data
$ s3-dist-cp --src s3://lanston-bigdata-final-exam/ads.csv --dest /user/Hadoop/ads/
```

2.4 2.4

Hive Query to Show Top 10 app'ids by CTRs:

```
literate
--hive query
SELECT
    app_id,
    impression,
    clicks,
```

```
    CTR
FROM (
    SELECT
        app_id,
        SUM(*) AS impression,
        SUM(CASE WHEN click = 1 THEN 1 ELSE 0 END) AS clicks,
        IF(SUM(*) > 0, SUM(CASE WHEN click = 1 THEN 1 ELSE 0 END) / SUM(*), 0) AS CTR
    FROM ads_lanston
    GROUP BY app_id
) a
WHERE impression >= 10
ORDER BY CTR DESC
LIMIT 10;
```

2.5 2.5

2.5.1 Pig Script to Repeat Query:

```
literate
ads = LOAD '/user/Hadoop/ads/ads.csv' USING PigStorage(',') AS (
    id: INT,
    click: INT,
    hour: CHARARRAY,
    C1: INT,
    banner_pos: INT,
    site_id: CHARARRAY,
    site_domain: CHARARRAY,
    site_category: CHARARRAY,
    app_id: CHARARRAY,
    app_domain: CHARARRAY,
    app_category: CHARARRAY,
    device_id: CHARARRAY,
    device_ip: CHARARRAY,
    device_model: CHARARRAY,
    device_type: INT,
    device_conn_type: INT,
    C14: INT,
```

```
    C15: INT,
    C16: INT,
    C17: INT,
    C18: INT,
    C19: INT,
    C20: INT,
    C21: INT
);
```

2.6 2.6

2.6.1 pySpark Script with Spark SQL Query:

```
literate
-- Group by app_id
grouped_ads = GROUP ads BY app_id;

-- Calculate impressions, clicks and CTR
aggregated_ads = FOREACH grouped_ads GENERATE
    group AS app_id,
    COUNT(ads) AS impressions,
    SUM(ads.click) AS clicks,
    (double)SUM(ads.click) / (double)COUNT(ads) AS CTR;

-- Filter for at least 10 impressions and sort by CTR
filtered_ads = FILTER aggregated_ads BY impressions >= 10;
sorted_ads = ORDER filtered_ads BY CTR DESC;

-- Limit to top 10
top_ads = LIMIT sorted_ads 10;

-- Output the results
DUMP top_ads;
```

2.7 Performance Analysis of Hive, Pig, and Spark

Based on the query execution times observed for Hive (13 seconds), Pig (28 seconds), and Spark (8 seconds), several insights can be drawn about the performance and efficiency of these big data processing frameworks.

- **Spark's Performance:** Spark's impressive speed, evidenced by the shortest query time, can be attributed to its advanced in-memory data processing capabilities. This allows Spark to efficiently handle iterative algorithms and interactive data mining, significantly outperforming traditional MapReduce tasks by reducing disk read/write operations.
- **Pig's Processing Time:** The longer processing time observed for Pig suggests that its execution strategy, which involves translating scripts into a series of MapReduce jobs, might not have been as optimized for this particular query as Spark's. The overhead associated with each MapReduce job in Pig could contribute to the longer execution times, particularly for complex transformations.
- **Hive's Middle Ground:** Hive's performance, while faster than Pig, is slower than Spark. Hive, like Pig, translates queries into MapReduce jobs, but its execution might be more optimized for certain batch processing tasks, leading to faster execution than Pig. However, it still lags behind Spark's in-memory processing.

These results underscore Spark's suitability for tasks requiring quick, iterative processing, particularly when working with memory-intensive operations. Meanwhile, Hive and Pig might be preferred for simpler batch processing tasks where the in-memory advantages of Spark are less critical. The choice of tool is heavily dependent on the specific use case, data size, complexity of operations, and the balance between speed and simplicity in development.

3 Section 3

3.1 Nobel Prize Winners Analysis

```
literate
from pyspark.sql import SparkSession

spark = SparkSession.builder \
        .appName("Nobel␣Prize␣Winners") \
        .getOrCreate()

file_path = "/content/bigdata-final/json_award.json"
df = spark.read.json(file_path)

df.createOrReplaceTempView("nobel_prize_winners")
from pyspark.sql.functions import explode, col

# Explode the laureates array
df_exploded = df.withColumn("laureate", explode("laureates"))

# Register the DataFrame as a temporary view
```

```
df_exploded.createOrReplaceTempView("exploded_laureates")

# Run the SQL query
result = spark.sql("""
    SELECT laureate.id, laureate.knownName.en as name, COUNT(*) as prize_count
    FROM exploded_laureates
    GROUP BY laureate.id, laureate.knownName.en
    HAVING COUNT(*) >= 2
""")

result.show()
```

3.2 Nobel Prize Word Analysis

```
literate
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, split, col, lower

# Create a Spark session
spark = SparkSession.builder \
            .appName("Nobel␣Prize␣Word␣Analysis") \
            .getOrCreate()

# Read the JSON file
file_path = "/content/bigdata-final/json_award.json"  # Adjust file path
df = spark.read.json(file_path)

# Explode the laureates array and register the DataFrame as a temporary view
df_exploded = df.withColumn("laureate", explode("laureates"))
df_exploded.createOrReplaceTempView("exploded_laureates")

# Run the SQL query to count word frequency
word_frequency = spark.sql("""
    SELECT word, COUNT(*) as frequency
    FROM (
        SELECT explode(split(lower(laureate.motivation.en), ' ')) as word
```

```
        FROM exploded_laureates
    )
    WHERE word <> ''
    GROUP BY word
    ORDER BY frequency DESC
    LIMIT 5
""")

word_frequency.show()
```

4 Question4

4.1 Utilizing the Dataset

To effectively utilize the dataset, the following data processing tools and their specific applications will be employed:

4.1.1 Python Libraries

The dataset will be processed using Python libraries, including Pandas and NumPy, to ensure data cleanliness and structure. These libraries provide robust capabilities for data manipulation, making them foundational tools for handling large datasets.

4.1.2 Visualization

Matplotlib and Seaborn will be utilized for data visualization. These libraries enable the creation of visual representations of the data, allowing for the exploration of initial data distributions and the identification of potential outliers or anomalies that could impact the modeling process.

4.2 Understanding the Customer Base

In order to gain insights into the customer base, the following data features will be analyzed:

4.2.1 Data Features

Variables such as ‘channelGrouping‘ and ‘browser‘ will be examined to identify distinct user segments. Engagement metrics, including ‘hits‘ and ‘pageviews‘, will be used to gain insights into user interactions with the Google Store. Understanding user behavior and paths to purchase is vital for shaping marketing and sales strategies.

4.3 Predicting Sales with the Gamma Hurdle Model

To predict sales effectively, the Gamma Hurdle Model will be applied. This model combines a logistic regression component to handle zero-revenue events and a Gamma regression component to model the magnitude of sales when purchases occur.

4.3.1 Model Structure

The Gamma Hurdle Model's structure is designed to address the unique characteristics of the data, where a small percentage of users contribute to the majority of sales, and a significant number of browsing sessions do not result in purchases.

4.3.2 Justification for the Gamma Hurdle Model

The choice of the Gamma Hurdle Model is justified by several factors:

- **Skewed Sales Data**: The model is well-suited for data with a 20/80 distribution, where a small fraction of customers generates most of the revenue.
- **Zero-Inflation**: The model's hurdle component is crucial for handling the high prevalence of zero sales in the dataset, a common challenge in retail analytics.
- **Revenue Prediction**: The Gamma distribution component of the model takes over once the hurdle of zero sales is crossed, enabling nuanced sales predictions.

4.3.3 Critical Reasoning

The selection of the Gamma Hurdle Model is strategic, considering the data's nature and distribution. It reflects an understanding of the customer behavior dichotomy, where many users browse but only a few make purchases that drive revenue. The model's dual approach aligns with this behavior, allowing for precise sales predictions that inform effective sales and marketing strategies.

In summary, this plan leverages robust data processing tools to prepare the dataset and employs a predictive model specifically chosen for its compatibility with the unique characteristics of the sales data. This approach ensures accurate and actionable insights for decision-making in sales and marketing strategies.

4.4 Challenges in Utilizing the Gamma Hurdle Model for Customer Targeting in Advertising

Utilizing the Gamma Hurdle Model for customer targeting in advertising introduces several key challenges that must be addressed:

1. **Sparse Data Interpretation**: The high prevalence of zero sales in the dataset can lead to challenges in correctly interpreting and classifying potential buyers, directly impacting the precision of targeting efforts.
2. **Model Interpretability**: The dual nature of the Gamma Hurdle Model, with its logistic regression and Gamma regression components, may complicate the extraction of straightforward insights from the model for making actionable advertising decisions.
3. **Behavioral Shifts**: Historical data may not accurately reflect current or future customer behavior, posing a risk to the relevance of the targeting strategy. Customer behavior can evolve over time.

4. **Selective Feature Relevance**: Incorrect feature selection or engineering could bias predictions and diminish the effectiveness of targeted advertising efforts. Careful feature selection is crucial.

5. **Segmentation Balance**: Achieving the right level of customer segmentation is critical. Both overly broad and overly narrow segmentation can result in suboptimal targeting, affecting advertising outcomes.

6. **Regulatory Compliance**: Adhering to data privacy laws and regulations is mandatory. Personalizing marketing efforts while ensuring compliance with privacy laws presents a significant challenge.

7. **Threshold Accuracy**: Setting an accurate threshold for sales predictions is crucial. Misjudgments in threshold selection can lead to missed opportunities or wasted advertising resources.

Addressing these challenges requires a comprehensive and thoughtful approach that combines technical expertise with a deep understanding of market dynamics and regulatory frameworks. Continuous model refinement, feature reassessment, and vigilance in monitoring and adapting to changing customer behavior are essential to maintain the accuracy and effectiveness of the targeting strategy.

4.5 Efficient Data Management and OLAP Star Schemas

4.5.1 Proposed Hive Database Schema

To efficiently manage and analyze the diverse data crucial for GStore's business operations, the following streamlined Hive database schema is proposed:

1. Customer Data Table ('customer˙data')

- **Focus**: Stores comprehensive customer-related information.
- **Key Columns**: 'customer˙id', 'email˙address', 'channel˙grouping', 'browser', 'device˙category', 'geographic˙details'.
- **Purpose**: Enables customer segmentation and behavior analysis.

2. Financial Data Table ('financial˙data')

- **Focus**: Contains transactional and financial performance data.
- **Key Columns**: 'transaction˙id', 'customer˙id', 'transaction˙revenue', 'sales˙statistics', 'costs'.
- **Purpose**: Essential for evaluating financial health and marketing ROI.

3. Operational Data Table ('operational˙data')

- **Focus**: Captures data on business operations.
- **Key Columns**: 'operation˙id', 'shipping˙details', 'CRM˙feedback', 'customer˙id'.
- **Purpose**: Helps in optimizing business processes and reducing operational costs.

4. Interaction Data Table ('interaction˙data')

- **Focus**: Records customer website interactions.
- **Key Columns**: 'session_id', 'customer_id', 'hits_summary', 'pageviews_summary'.
- **Purpose**: Provides insights into customer engagement and website performance.

Critical Reasoning:

- This schema is designed to align with GStore's analytical needs, segregating data into distinct yet interconnected domains for targeted analysis.
- Aggregated columns in the interaction data indicate pre-processing to summarize user activities, enhancing query efficiency.
- The structure supports a multi-dimensional view of the business, from customer behavior to financial performance and operational efficiency, allowing for holistic and nuanced analyses.

In summary, the schema balances comprehensiveness with practicality, ensuring that data is not only stored efficiently but is also ready for complex, multidimensional queries that drive business decisions.

4.5.2 OLAP Star Schemas for Customer Experience Improvement

For improving customer experience, three OLAP (Online Analytical Processing) star schemas are proposed, each with a unique focus:

1. Personalization Star Schema

- **Fact Table**: Sales Transactions
- **Key Measures**: Sale Amount, Number of Transactions
- **Dimension Tables**: - Customers: Customer ID, Demographics, Purchase History - Products: Product ID, Category, Price - Time: Date, Month, Year - Sales Channel: Channel ID, Channel Type
- **Purpose**: Tailoring the customer experience through personalized product recommendations and targeted marketing based on individual customer profiles.

2. Understanding Customer Behavior Star Schema

- **Fact Table**: Customer Interactions
- **Key Measures**: Session Duration, Page Views, Click-Through Rates
- **Dimension Tables**: - Customers: Customer ID, Behavioral Segments, Engagement Level - Web Pages: Page ID, Content Type, URL - Time: Date, Time of Day, Weekday/Weekend - Device: Device Type, Operating System, Browser
- **Purpose**: Analyzing customer online behavior to gain insights into how customers interact with the website, which can inform UI/UX improvements and content strategy.

3. Predicting Future Trends Star Schema

- **Fact Table**: Market Trends
- **Key Measures**: Trend Score, Popularity Index, Growth Rate
- **Dimension Tables**: - Time: Date, Month, Quarter, Year - Product Categories: Category ID, Category Name, Department - Customer Demographics: Age Group, Income Level, Region - Sales Data: Sales Amount, Number of Transactions, Average Purchase Value
- **Purpose**: Forecasting future market trends to help the organization stay ahead in product development and inventory management.

These schemas are designed to leverage multidimensional data analysis, enabling GStore to derive actionable insights and make informed decisions to enhance customer experience and drive revenue growth.

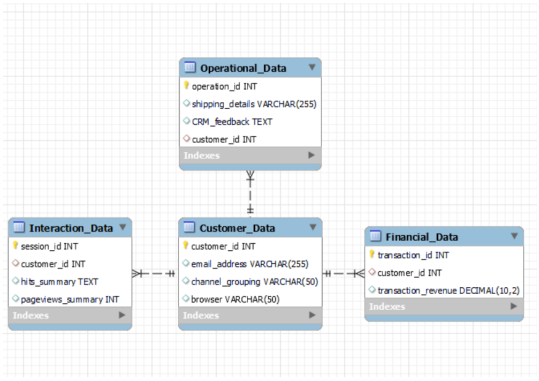

Figure 6: ER model

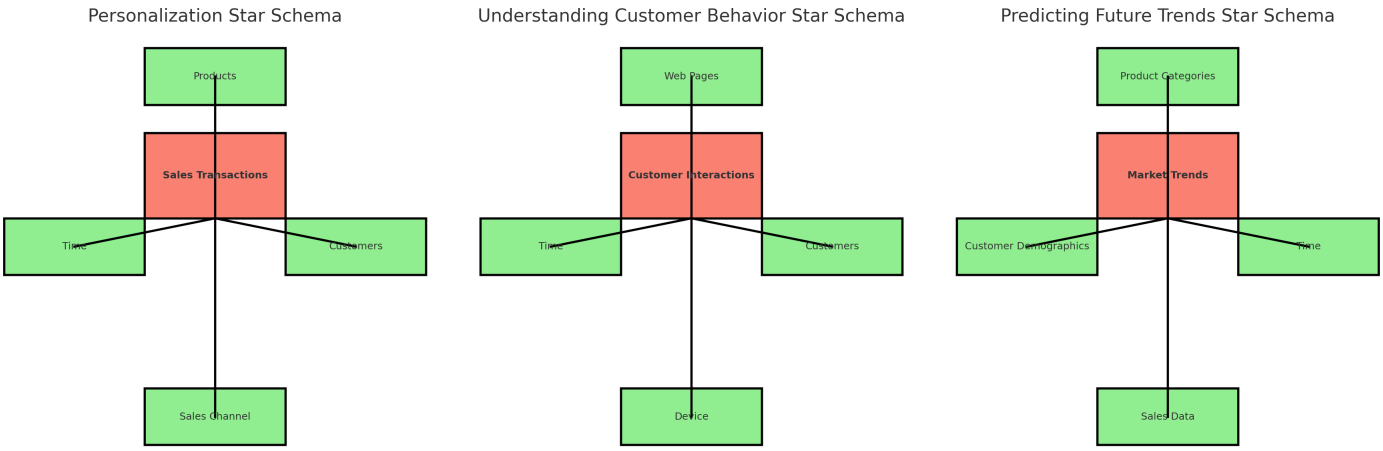

Figure 7: Start Schema